WHICH ARCHITECTURE IS BETTER SUITED FOR MATRIX-FREE FINITE-ELEMENT ALGORITHMS: INTEL SKYLAKE OR NVIDIA VOLTA?

MARTIN KRONBICHLER, MOMME ALLALEN, MARTIN OHLERICH, WOLFGANG A. WALL
KRONBICHLER@LNM.MW.TUM.DE
TECHNICAL UNIVERSITY OF MUNICH, LEIBNIZ SUPERCOMPUTING CENTRE

MATRIX-FREE ALGORITHM LAYOUT

Matrix-free algorithm in finite element programs exchanges **matrix-vector product** in matrix-based scheme

$$\begin{cases} A = \sum_{K\in\{\text{cells}\}} P_K^{\mathrm{T}} A_K P_K \text{ (with assembly)} \\ v = Au \text{ (sparse mat-vec within iterative solver)} \end{cases}$$

by evaluation of integrals **within the iterative solver**:

$$v = \sum_{K\in\{\text{cells}\}} P_K^{\mathrm{T}} A_K \left(P_K u\right)$$

Matrix-free algorithm:

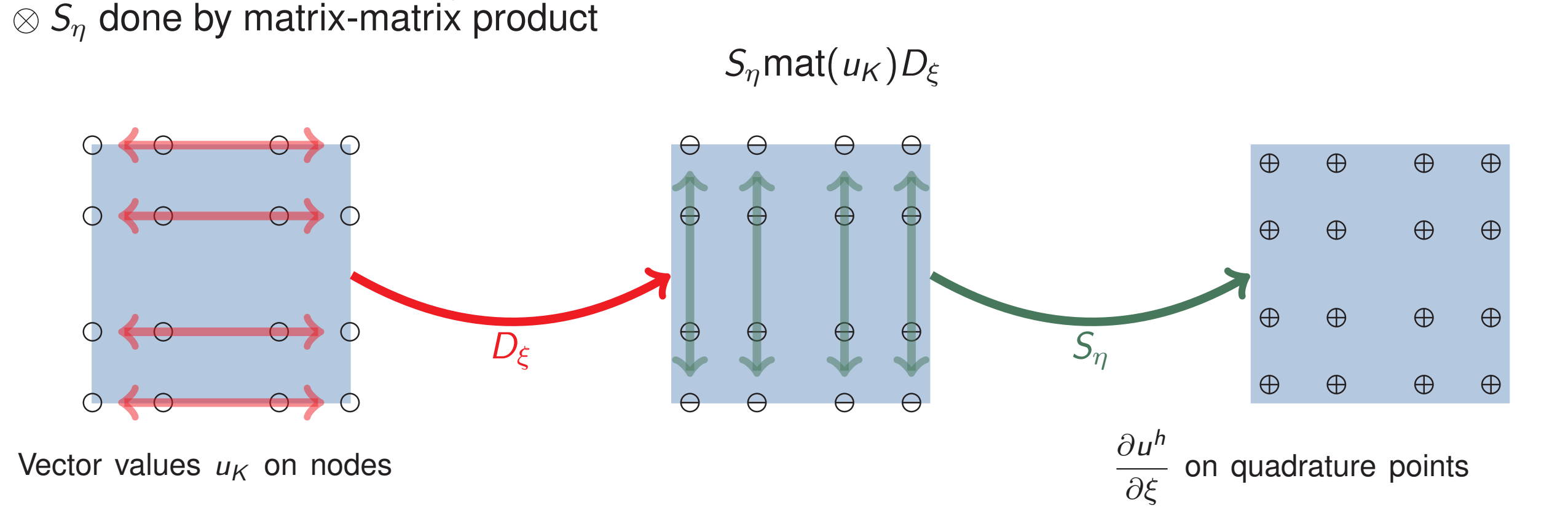
- $v = 0$
- loop over cells
  - (i) Extract local vector values on cell: $u^{(K)} = P_K u$
  - (ii) Apply operation locally on cell: $v^{(K)} = A^{(K)} u^{(K)}$ (**without** forming $A^{(K)}$)
  - (iii) Sum results from (ii) into the global solution vector: $v = v + P_K^{\mathrm{T}} v^{(K)}$

MATRIX-FREE CELL OPERATION FOR LAPLACIAN

- Weak form: $(\nabla\varphi_i, \nabla u_h)_\Omega$ represented in matrix-free way
- Approximation on each cell K with Gaussian quadrature on q^d points in d dimensions:

$$\begin{aligned}(\nabla\varphi_i, \nabla u_h)_K &= \int_{K_{\text{unit}}} \left(\mathcal{J}^{(K)}(\xi)^{-\mathrm{T}} \nabla_\xi \varphi(\xi)\right) \cdot \left(\mathcal{J}^{(K)}(\xi)^{-\mathrm{T}} u_h^{(K)}(\xi)\right) \det(\mathcal{J}^{(K)}(\xi))\, \mathrm{d}\xi \\ &\approx \sum_{r=1}^{q^d} \left(\mathcal{J}^{(K)}(\xi_r)^{-\mathrm{T}} \nabla\varphi_i(\xi_r)\right) \cdot \left(\mathcal{J}^{(K)}(\xi)^{-\mathrm{T}} u_h^{(K)}(\xi_r)\right) \det(\mathcal{J}^{(K)}(\xi_r)) w_r\end{aligned}$$

- **Efficient computation of integrals:** Sum factorization on quadrilaterals/hexahedra through `deal.II` finite element library `www.dealii.org` [1, 4, 6, 7]
- Sum factorization used for interpolation kernels $\nabla u_h^{(K)}(\xi_r) = \sum_{j=1}^{(p+1)^d} \nabla\varphi_j(\xi_r) u_j^{(K)}$ and summation over quadrature points in r
- Example for evaluation of $\frac{\partial u}{\partial \xi}$ in all quadrature points, given node values u_K with interpolation matrix $D_\xi \otimes S_\eta$ done by matrix-matrix product


$S_\eta \mathrm{mat}(u_K) D_\xi$
D_ξ
S_η
Vector values u_K on nodes
$\frac{\partial u^h}{\partial \xi}$ on quadrature points

HARDWARE SETUP

- Intel Skylake: 2-socket Xeon Scalable Platinum 8168, 2×24 cores at 2.5 GHz (max AVX-512 frequency)
- Intel Broadwell: 2-socket Xeon E5-2698 v4, 2×20 cores at 2.2 GHz
- Intel KNL: Xeon Phi 7210, 64 cores at 1.1 GHz (max AVX-512 frequency)
- NVIDIA Volta V100
- NVIDIA Pascal P100

REFERENCES

[1] G. Alzetta, D. Arndt, W. Bangerth, V. Boddu, B. Brands, D. Davydov, R. Gassmoeller, T. Heister, L. Heltai, K. Kormann, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, D. Wells (2018), The `deal.II` library, version 9.0. *Journal of Numerical Mathematics*. `doi:10.1515/jnma-2018-0054`.

[2] N. Fehn, W. A. Wall, M. Kronbichler (2018), Efficiency of high-performance discontinuous Galerkin spectral element methods for under-resolved turbulent incompressible flows. *International Journal for Numerical Methods in Fluids* 88(1):32–54. `doi:10.1002/fld.4511`.

[3] P. Fischer, Tz. Kolev, M. Min, V. Dobrev, et al. (2018), CEED bake-off problems for matrix-free operator evaluation, `http://ceed.exascaleproject.org/bps/`.

[4] M. Kronbichler, K. Kormann (2012), A generic interface for parallel cell-based finite element operator application, *Computers & Fluids* 63:135–147. `doi:10.1016/j.compfluid.2012.04.012`.

[5] M. Kronbichler, K. Kormann (2017), Fast matrix-free evaluation of discontinuous Galerkin finite element operators, arXiv preprint arXiv:1711.03590.

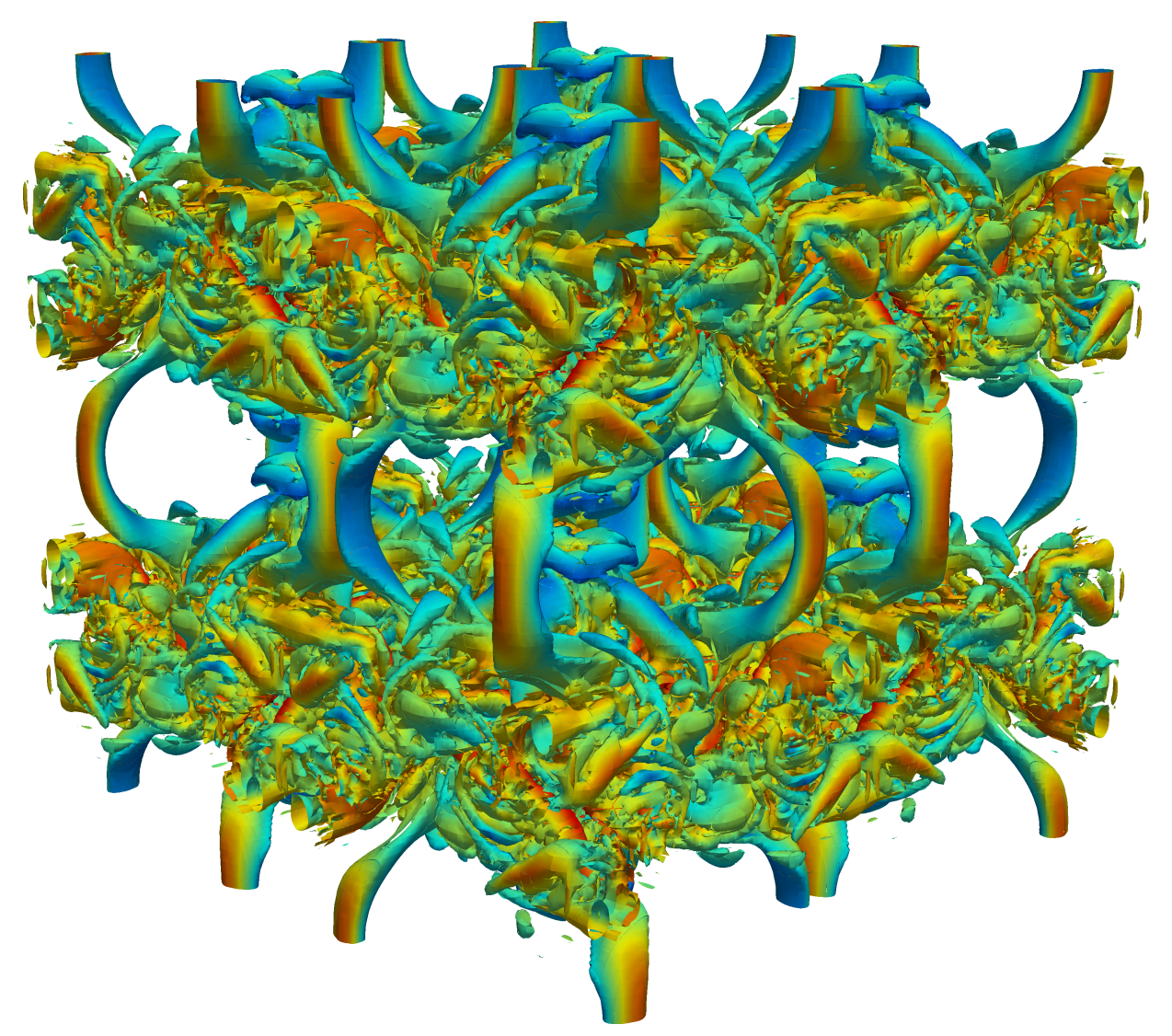
[6] M. Kronbichler, K. Kormann, I. Pasichnyk, M. Allalen (2017), Fast Matrix-Free Discontinuous Galerkin Kernels on Modern Computer Architectures, in J. Kunkel , R. Yokota, P. Balaji, D. Keyes (eds): *High Performance Computing. ISC 2017*. Lecture Notes in Computer Science, vol 10266, pp. 237–255, `doi:10.1007/978-3-319-58667-0_13`.

[7] M. Kronbichler, W. A. Wall (2018), A performance comparison of continuous and discontinuous Galerkin methods with fast multigrid solvers, *SIAM J. Sci. Comput.*, in press, `doi:10.1137/16M110455X`.

[8] K. Ljungkvist, M. Kronbichler (2017), Multigrid for Matrix-Free Finite Element Computations on Graphics Processors, Technical Report 2017-006, Department of Information Technology, Uppsala University.

SUMMARY

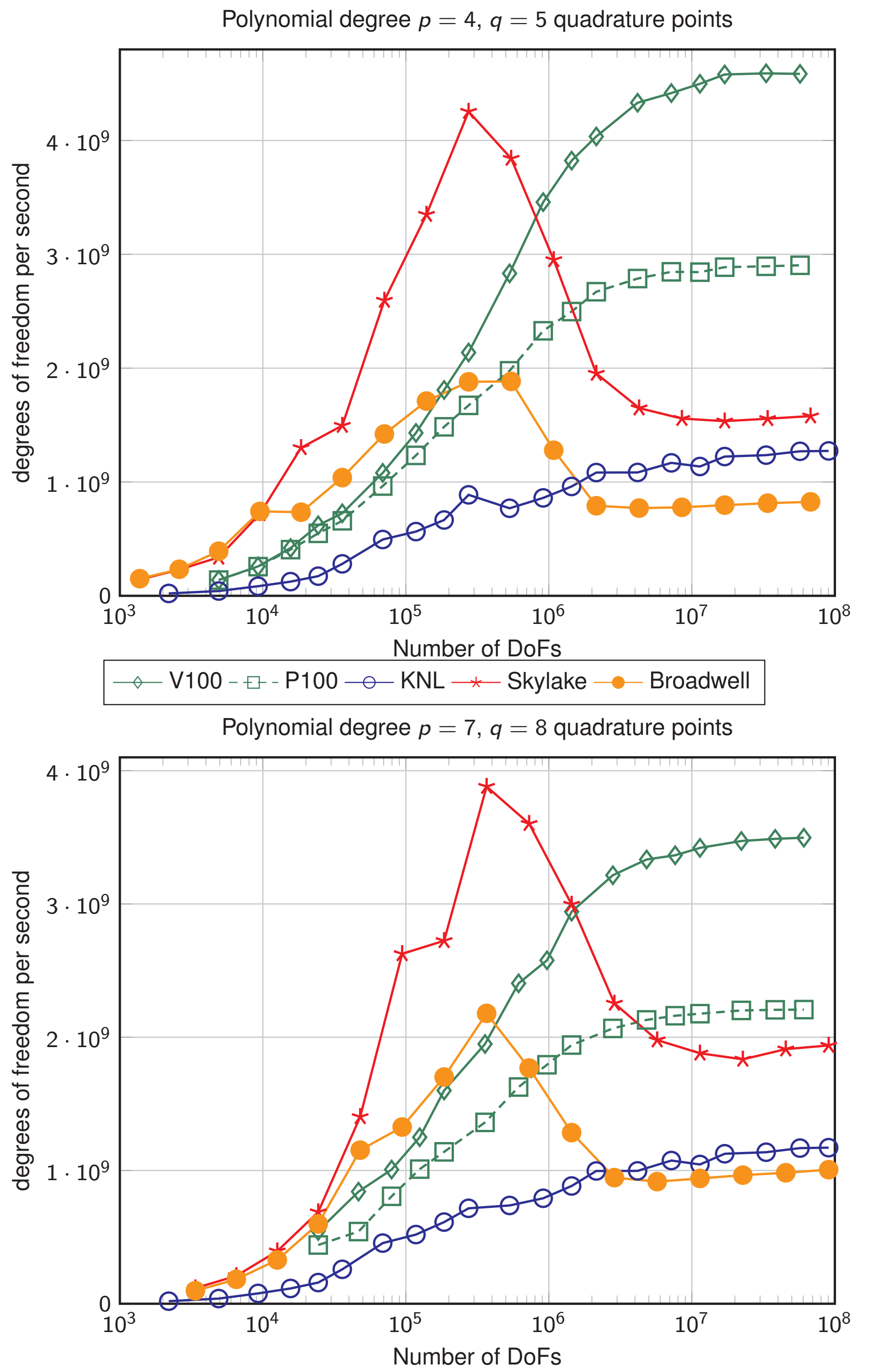
- Performance evaluation of matrix-free finite element kernels from `deal.II` library (`www.dealii.org`) on Intel Broadwell, Intel KNL, Intel Skylake, NVIDIA Pascal, and NVIDIA Volta
- Analysis of matrix-free operator evaluation as proxy for application performance in fluid dynamics [2]
- Volta 1.6× faster than Pascal; Volta 2×–3× faster than Skylake for large sizes; from L2/L3 cache, Skylake reaches similar performance as Volta
- For large problem sizes, CPUs suffer from relatively low memory bandwidth (Skylake theoretical performance: 255 GB/s, Volta theoretical performance: 900 GB/s)
- KNL not competitive due to mixture of heavy arithmetic in sum factorization and memory transfer in quadrature and missing hardware prefetching; only 200 GB/s
- NVLink communication on multiple GPUs with MPI-like setup: explicitly send ghost data, overlapped with computations → good in weak scaling setup, but difficult to maintain low latency of single-GPU case → further research necessary → CPU advantageous in latency-sensitive regime



Application background: Simulation of 3D Taylor–Green vortex at $\mathrm{Re} = 1600$, flow field visualized by Q-criterion [2]

NODE-LEVEL PERFORMANCE

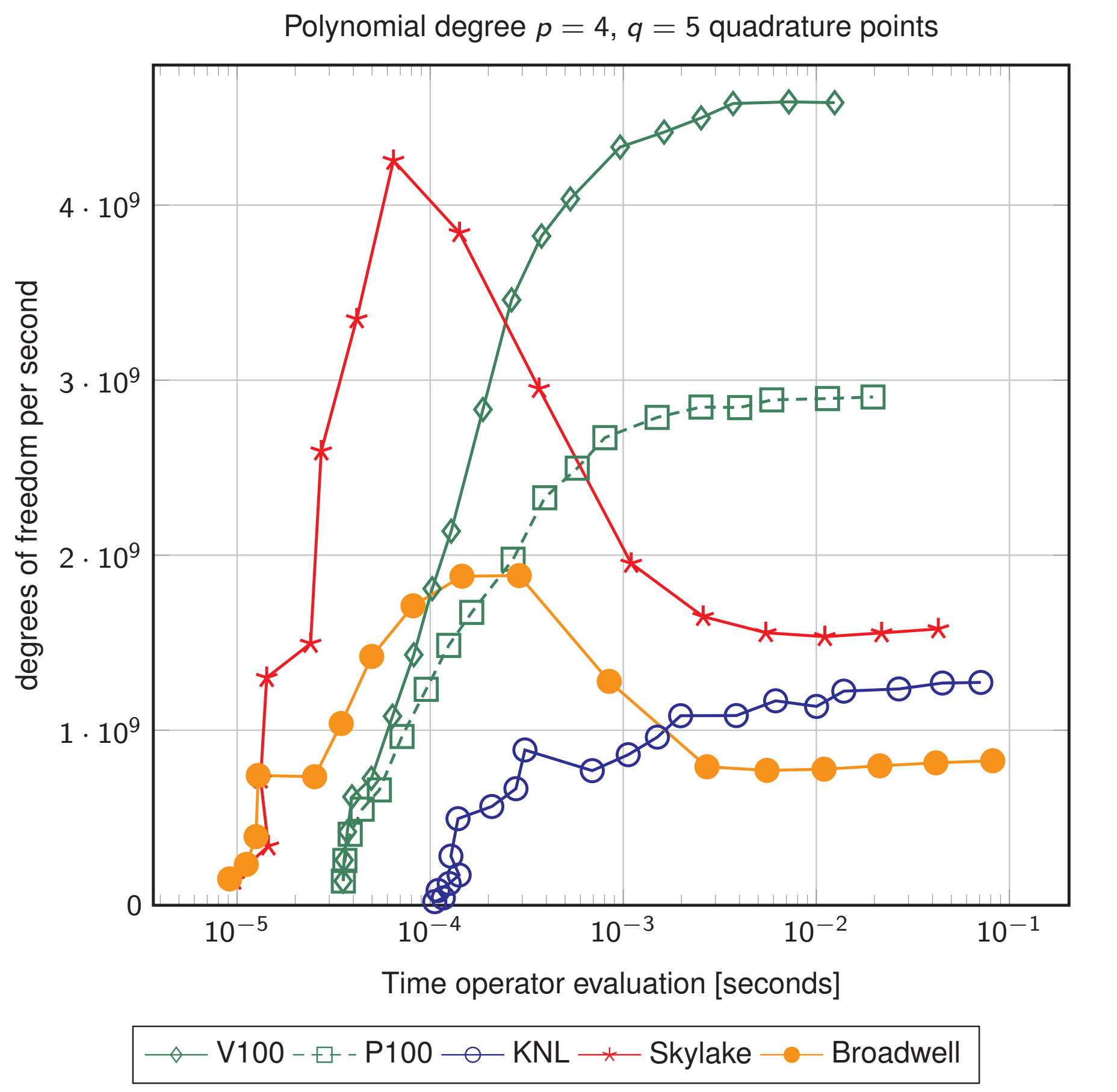
- Analyze the performance on kernel similar to CEED bake-off problems [3]
- Setup: 16 to 10^6 mesh cells, deformed geometry
- MPI parallelization of CPU codes, parallel CUDA kernels on GPUs according to [8]:
  - Loop over cells parallelized, use atomics to avoid race conditions
  - One thread per local DoF on elements
- Choose polynomial degree p, Gaussian quadrature with $q = p+1$ quadrature points (similar to CEED BP5 problems, but full Gaussian quadrature rather than Gauss–Lobatto)
- Merged coefficient tensor $\mathcal{J}^{-1}\mathcal{J}^{-T}\det(\mathcal{J})w_q$ stored in each quadrature point, i.e., 6×8 bytes per quadrature point
- Measure performance of matrix-vector product only (BK5), repeat 100 times
- CPU gains speed more quickly on small problem sizes, reaching excellent performance for 10^5 DoFs
- **Skylake** 2× **faster than Volta for 300k DoFs**
- Up to 10^6 DoFs, all data fits into L2/L3 caches on Broadwell and Skylake → high throughput
- Skylake twice as fast as Broadwell from caches due to AVX-512 (implementation uses 8-wide vectorization over several elements according to [5]) and more cores (48 vs 40)
- CPU performance drops significantly once access must go to main memory
- Measured performance at 50 million DoFs and $p = 4$: 115 GB/s on Broadwell, 220 GB/s on Skylake, 690 GB/s on Volta
- **Volta** more than **2× faster than Skylake at large sizes**
- **Skylake from cache** reaches approximately **same throughput as Volta** (served from high-bandwidth memory)
- Volta consistently 1.6 times faster than Pascal on most benchmarks
- KNL not really competitive – cannot fully exploit high-bandwidth memory due to mixture of arithmetic heavy parts and memory transfer (missing prefetching)



THROUGHPUT VERSUS LATENCY

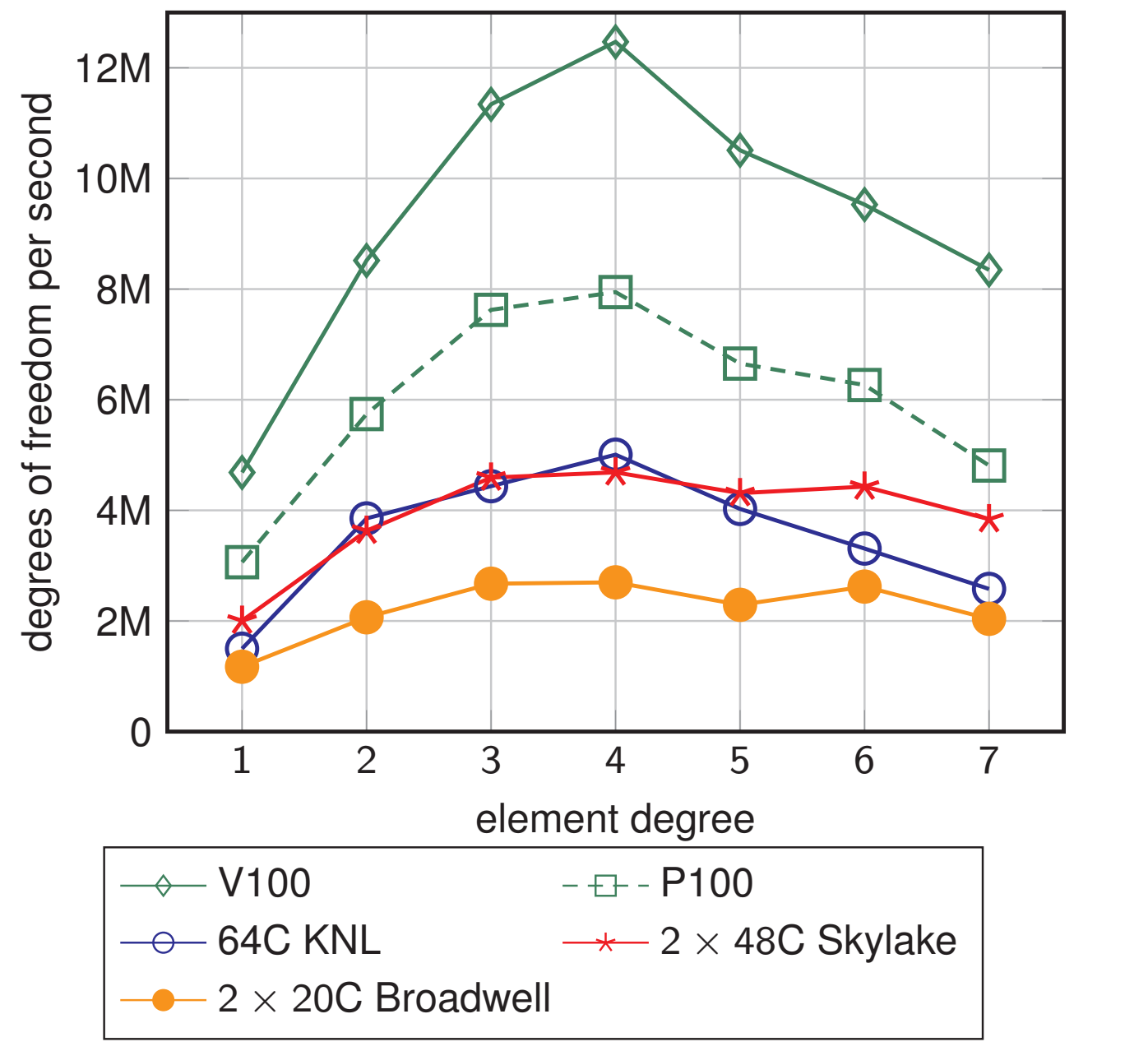
Analysis of latency of the various architectures: Plot throughput over the absolute time for matrix-vector product

- Skylake and Broadwell reach good throughput down to $3 \cdot 10^{-5}$ seconds
- Volta and Pascal only efficient above $5 \cdot 10^{-4}$ seconds
- CPU architectures benefit from fast caches
- **CPU architectures more promising for strong scaling** of applications with multigrid components where time per operator evaluation for small sizes is critical
- KNL worst architecture in this metric and hampered due to
  - MPI-only parallelization
  - many relatively slow cores
  - vectorization over several cells



MULTIGRID APPLICATION PERFORMANCE

- Laplacian with variable coefficient $a(\mathbf{x}) = 1 + 10^6 \prod_{e=1}^{d} \cos(2\pi x_e + 0.1e)$, analytic solution $u(\mathbf{x}) = \sin\left(\pi(x+y)\right)$
- 3D shell geometry, high-order curved elements
- Conjugate gradient iterative solver preconditioned by geometric multigrid based on `deal.II` infrastructure [7, 8], ∼ 15 iterations
- Polynomial Chebyshev smoother of degree 5 (=5 mat-vec) for pre- and post-smoothing
- Multigrid cycle done in single precision, outer CG in double precision → leverages 2× higher throughput of `float`
- Multigrid solver is central component in incompressible flow solver according to [2]

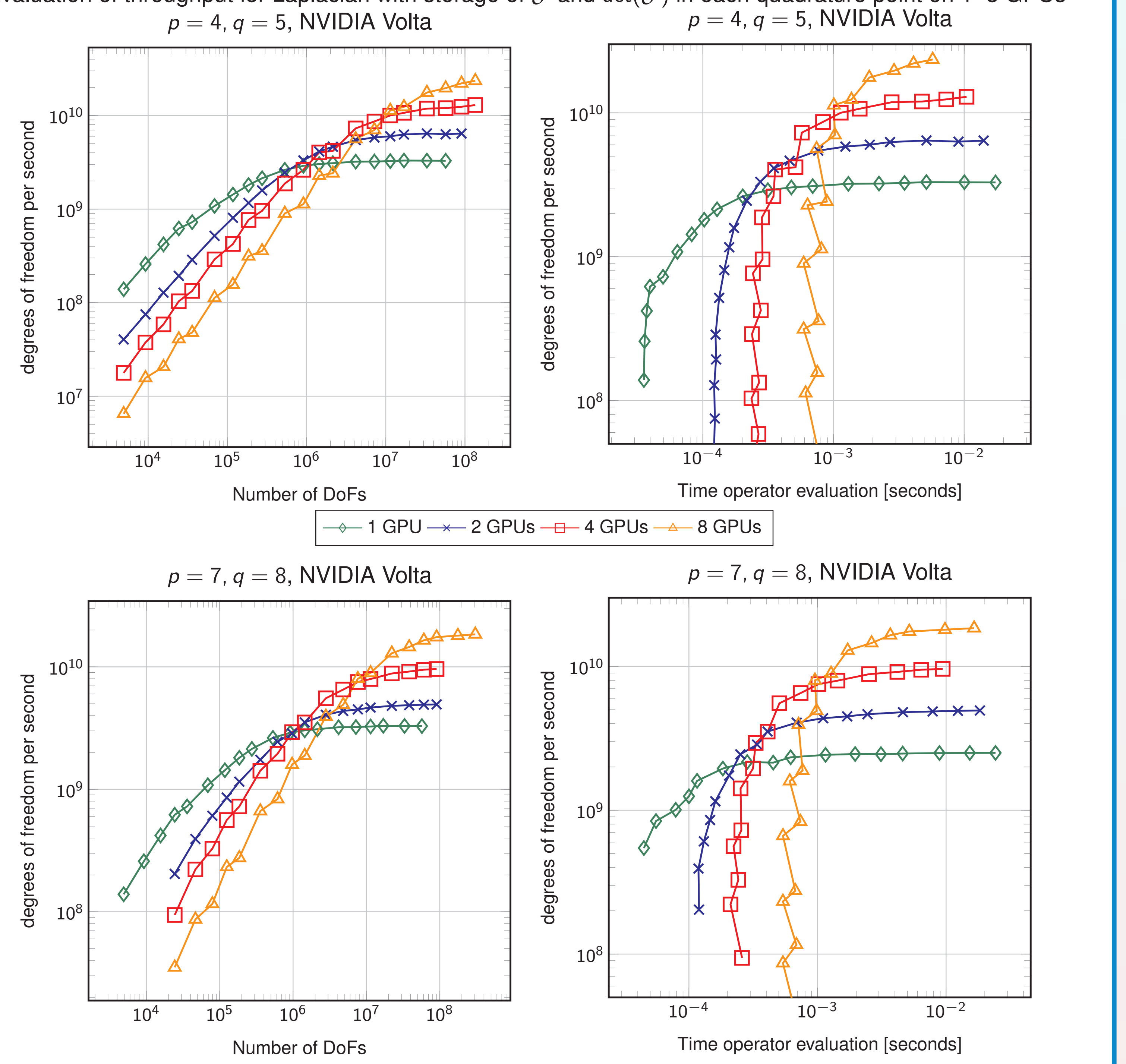


- Matrix-free evaluation several times faster than matrix-based algorithms [7]
- Due to the deformed geometry, this test case is almost completely **memory bandwidth bound**
- NVIDIA Volta V100 reaches up to 650 GB/s, Skylake 220 GB/s
- Coarser grid levels faster on CPUs than on KNL and V100/P100

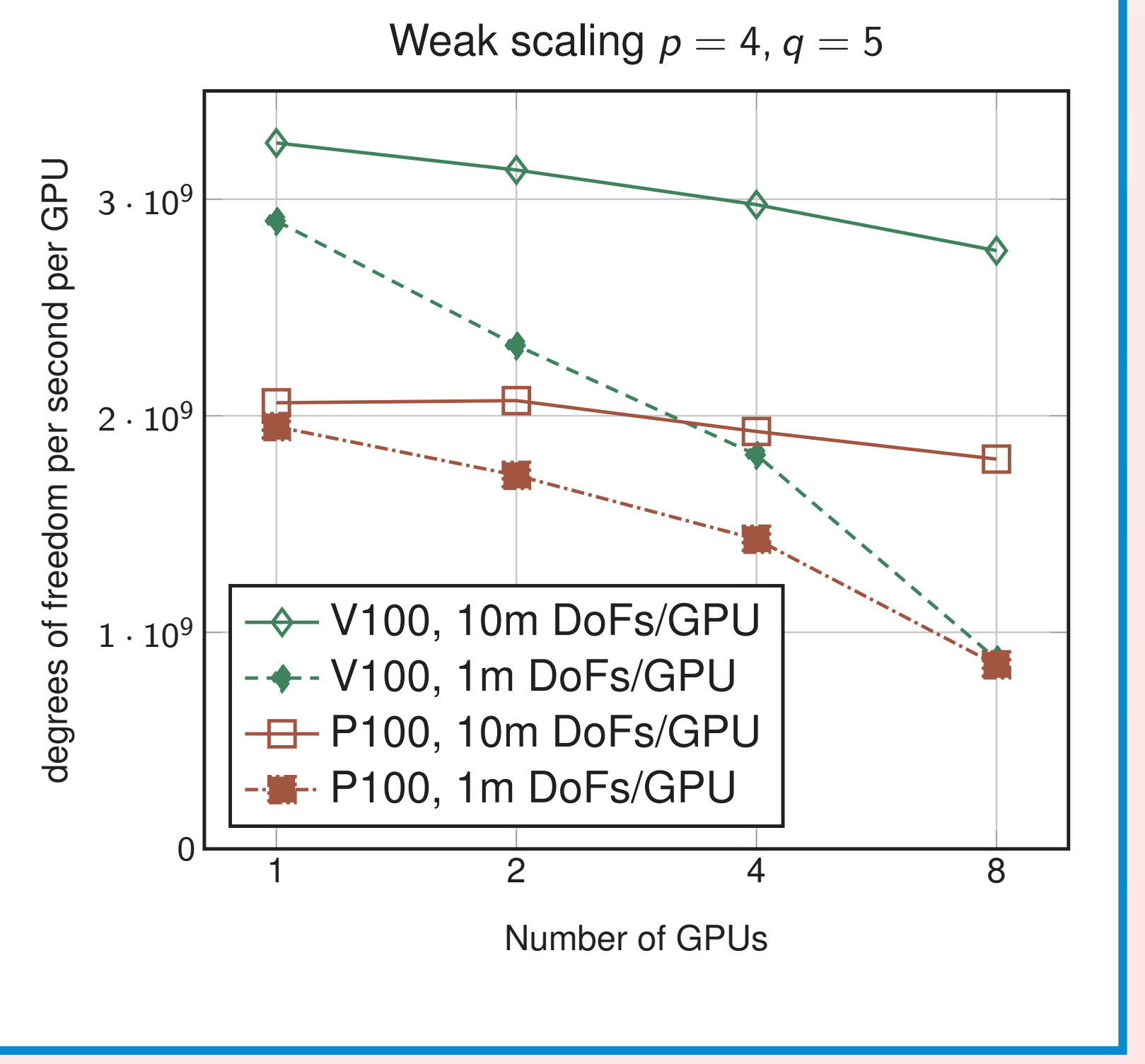
MULTI-GPU ANALYSIS ON NVIDIA DGX-1

- Parallelization to multiple GPUs on NVIDIA DGX-1 (for P100 and V100)
- MPI-like setup with separate domains for each GPU, data exchange between GPUs via NVLink/NVSwitch protocol with `cudaMemcpyPeerAsync`
- Topology of NVLink reflected in domain decomposition → large data exchange between GPUs with direct link

Evaluation of throughput for Laplacian with storage of $\mathcal{J}$ and $\det(\mathcal{J})$ in each quadrature point on 1–8 GPUs



- Multi-GPU setup provides good speedup for large problem sizes
- Latency severely impacted in multi-GPU setup: loss of around a factor of 10 when going from 1 to 8 GPUs, cannot go below 10^{-3}s on 8 GPUs!
- Almost ideal weak scaling for 10m DoFs per GPU
- Bad scaling for 1m DoFs per GPU
- **Cross-GPU communication** is a **serious bottleneck** for latency-sensitive applications
- In current implementation, multi-GPU scales worse than multi-node CPU codes presented in [7] which can reach $2 \cdot 10^{-4}$s
- Further research necessary to speed up multi-GPU case
  - Detailed multi-GPU performance analysis outstanding
  - Is there potential for more overlap of communication and computation?
  - Do we need to merge operations at a higher level between several matrix-vector products?


Weak scaling $p = 4, q = 5$
degrees of freedom per second per GPU
$1 \cdot 10^9$
$2 \cdot 10^9$
$3 \cdot 10^9$
Number of GPUs
V100, 10m DoFs/GPU
V100, 1m DoFs/GPU
P100, 10m DoFs/GPU
P100, 1m DoFs/GPU